Truthful vs. Deceptive Hotel Reviews

MP2 - Natural Language Report - G37

Guilherme Pereira
Instituto Superior Técnico
Lisbon, Portugal
guilhermecnpereira@tecnico.ulisboa.pt

Miguel Belbute
Instituto Superior Técnico
Lisbon, Portugal
miguel.belbute@tecnico.ulisboa.pt

1 INTRODUCTION

To address the task of classifying hotel reviews which were labeled with regards to truthfulness (TRUTHFUL vs. DECEPTIVE) and polarity (POSITIVE vs. NEGATIVE), resulting in four possible labels (TRUTHFULPOSITIVE, TRUTHFULNEGATIVE, DECEPTIVEPOSITIVE and DECEPTIVENEGATIVE), multiple models were developed in Python, being further explored in the following sections.

2 MODELS

Multiple **Machine Learning** (ML) classifier model were applied: Multinomial Naïve Bayes (MNB), Logistic Regression (LR), Support Vector Machine (SVM) and Stochastic Gradient Descent (SGD). The general pipeline for these models included data preprocessing, vectorization, training and evaluation. Moreover, two **Deep Learning** (DL) approaches were also considered: Temporal Convolutional Network (TCN) using two different word embedding modes - a static word embedding using pre-trained Word2Vec mode and a random mode; and Bidirectional Enconder Representations from Transformers (BERT). More specifically, **five implementations** were developed. The first contains and runs all the mentioned ML models to identify a clear baseline to improve upon. The second proceeds to fine-tune the best model from the first implementation. The third implementation generates the final labels for the test run with the best achieved model. The fourth and fifth contain and run the DL models.

2.1 Machine Learning

2.1.1 Preprocessing. The preprocessing was inspired by multiple applications found online (why reinvent the wheel?), involving word tokenization, stop word removal, stemming and detokenization [1]. In the first implementation the *NLTK*'s word tokenizer (NLTK is a Natural Language Processing Python library) and the Porter stemmer were employed for all the models [3]. However, the second implementation explored multiple tokenizers (e.g.: *NLTK*'s, *Regexp*'s, *Treebank*'s, *Whitespace*'s, *WordPunct*'s and *TokTok*'s) and stemmers (e.g.: *Porter*'s, *Lancaster*'s, *Snowball*'s and *Regexp*'s). Regarding stop word removal, various options were also analysed in detail since this process can heavily impact the performance of a text classification task. After examining the default stop words pertaining to *scikit-learn* (a ML Python library) and *NLTK*, a decision was made to edit these lists [6] [3]. Considering one of the goals is to determine the polarity of the reviews, we opted to exclude from the stop words "fingerprints" belonging to a communication concept entitled **negative language** [4]. This type of communication, often used when a person is plagued with negative emotions, is characterised by sentences in the negative form (e.g.: *"I don't recommend..."*) or passive-aggressive expressions (e.g.: *"Sure, it was fine but..."*). Therefore, in some tests, we decided to exclude words such as *"but"*, *"no"*, *"not"*, *"couldn't"*, *"never"* and *"however"* to view the impact on the overall performance.

2.1.2 Models' Implementation. Pipelines were originated for each classifier, being formed by two components: a vectorizer (we mostly utilized the TF-IDF vectorizer), which converts textual data into a numerical format, and the classifier itself (SVM, SGD, LR, and so on)[6].

2.2 Deep Learning

2.2.1 Preprocessing. For preprocessing data in DL, we found that just using a tokenizer was enough, as it did minimal text cleaning as to not damage each model's ability to learn [7][2]. For TCN, we utilized a tokenizer provided by the preprocessing text library within the *Keras* framework. Conversely, for BERT, we leveraged a BertTokenizer from the transformer library developed by *HuggingFace*, which is purposefully designed for data preprocessing tailored to BERT applications.

2.2.2 Models' Implementation. We chose to implement a variant of the TCN embedding random model described in a thesis by Raihan et al. [7]. The original model in the referenced paper employed a *sigmoid* activation function in the final layer to accommodate two distinct classes. To suit our requirement for four distinct classes, we modified the activation function to *softmax*. For the same reason, the loss function was also changed from *binary cross entropy* to *categorical cross entropy*. From the same study conducted by Raihan et al. [7], we adapted the TCN Word2Vec static model to accommodate four distinct classes, employing the same modifications mentioned before. Concerning BERT, we followed the approach outlined in a tutorial available on the *Into Deep Learning*'s website and in this case we did not need to make any alterations to the model [2].

3 EXPERIMENTAL SETUP AND RESULTS

The provided dataset, containing 1400 examples of review-label pairs, was split to originate a **training set - 90% of the original data** - and a **development set - the remaining 10%**. The latter was used to evaluate and fine-tune the various models during training. Before each training run, **the data was randomly shuffled** while still maintaining the aforementioned proportions. Finally, the provided test set - 200 reviews without the "gold labels" - was utilized to perform the final evaluation.

NL Project 2, IST, 2023, Lisbon, Portugal
© 2023 Association for Computing Machinery.
https://doi.org/

To evaluate the models' performances, we focused on **accuracy** as the primary evaluation metric. In determining this metric, we compared the predicted labels with the correct ones from the development sets. Recognizing the potential for significant variance in performance due to arbitrary partitions, we performed a **cross-validation** computation with 5 "folds" (or smaller sets), which provided us with the **mean accuracy** and its corresponding **standard deviation**. Although not directly evaluating the model, one should note how the suggested implementations also display the **confusion matrices** related to each run and how the final model prints out the **incorrect predictions**, facilitating a more in-depth analysis of the results.

With regards to parameters, on the first implementation only the **default values** were applied to the pipeline (these parameters can be consulted in the corresponding documentation) [6]. On the second implementation, though, a **grid search** was performed on a set of defined parameters to fine-tune and improve the overall accuracy of the model which had displayed the best performance so far. In the realm of DL models, no extensive parameter exploration was conducted. Consequently, the models were evaluated using their default parameters as provided in the original implementations[7][2].

For the ML models, the "default" TF-IDF + SVM pipeline achieved the highest accuracy at 81.36% on the first implementation. Subsequently, we chose to further analyze TF-IDF + SVM - Figure 1 illustrates the impact of different stop words on both the default and a fine-tuned version of the model. Notably, the **fine-tuned model without stop words** outperformed all others, achieving an **average score of 85.50%** - Figure 2 (both images were obtained with *NLP-Telescope* [5]). Considering the DL models: **TCN embedding random** achieved **69%**, **TCN Word2Vec static** reached **71.17%**, and the **BERT model** scored **74%**.

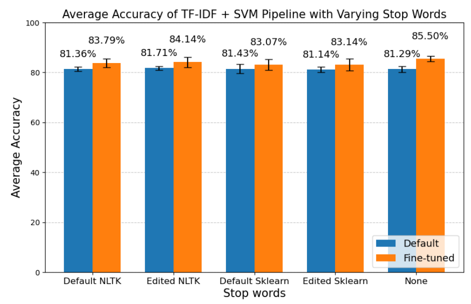

Figure 1: TF-IDF and SVM pipeline with default and fine-tuned parameters - average accuracy and standard deviation with varying stop words.

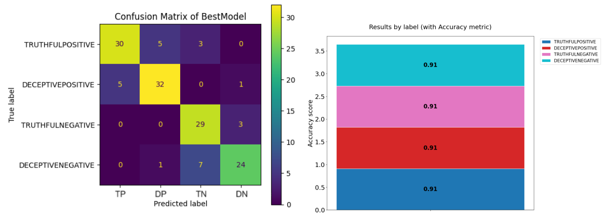

Figure 2: Fine-tuned TF-IDF + SVM - confusion matrix and accuracy per label.

4 DISCUSSION

Regarding the stop words' analysis (Figure 1), despite one verifying mostly positive variations on the average accuracy with the edited lists (especially regarding *NLTK*: 81.36% to 81.71% and 83.79% to 84.14%), considering the standard deviation identified, **one cannot confirm nor deny the impact of eliminating "negative communication" from the stop words**. This result can, however, be influenced by the biaxial nature of this classification task: aside from determining polarity, we also had to consider truthfulness, whose correlation to negative language remains unexplored.

Figure 2's confusion matrix showcases how **the fine-tuned model is greatly accurate in polarity classification**, with **the most frequent errors in all labels corresponding to mislabelling on the truthfulness scale** - TRUTHFUL*POSITIVE* is most often mistaken for DECEPTIVE*POSITIVE*, TRUTHFUL*NEGATIVE* is most frequently mistaken for DECEPTIVE*NEGATIVE* and so on. A good example of this is review 690 which, due to its unusual punctuation and structure (e.g.: *",this is a very good place.amazing"*) often induces the model to mistake it for DECEPTIVE when it is TRUTHFUL. Contrarily, when the predicted polarity is incorrect its often due to a review containing a mixture of good and bad adjectives (e.g.: review 887) or a review whose classification contradicts its content (e.g.: review 1237 contains *"Will definitely stay there again!"* yet is classified as negative).

Nonetheless, the **final model**, which outperformed all other models - **fine-tuned TF-IDF + SVM without stop word removal** - managed to achieve a satisfactory 91% accuracy per labels (Figure 2)! The underperforming of the DL models is possibly due to their large size and small data set, which poses an **overfitting risk**. ML models allowed for quick and convenient parameter tuning while DL models were time-consuming and computationally expensive, making parameter exploration challenging.

5 FUTURE WORK

To gain deeper insights into the influence of stop words associated with negative language [4], a specialized investigation focused on their impact solely regarding polarity classification could be conducted. Furthermore, future research endeavors could concentrate on refining our DL approaches, expanding the dataset, or simplifying network architecture by reducing the number of layers and their complexity.

REFERENCES

[1] Ander Fernández Jauregui. 2022. Naive Bayes in python. (Aug 2022). https://anderfernandez.com/en/blog/naive-bayes-in-python/

[2] Into Deep Learning. 2022. Multiclass text classification using bert. (Jun 2022). https://www.intodeeplearning.com/bert-multiclass-text-classification/

[3] Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. (2002). https://doi.org/10.48550/ARXIV.CS/0205028

[4] Rod Matthews. 2021. Negative language can have a big impact. (Sep 2021). https://www.rodmatthews.com.au/negative-language-can-have-a-big-impact/

[5] Rita Oliveira. 2023. NLP Telescope - a Hugging Face Space by RitaTO. (Sep 2023). https://huggingface.co/spaces/RitaTO/NLP-Telescope

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[7] Diardano Raihan. 2021. Deep learning techniques for text classification. (Apr 2021). https://towardsdatascience.com/deep-learning-techniques-for-text-classification-78d9dc40bf7c